INTERCEPT+: SDN Support for Live Migration-based Honeypots

Ayumu Hirata[*], Daisuke Miyamoto[†], Masaya Nakayama[†], Hiroshi Esaki[*]
[*] Graduate School of Engineering
The University of Tokyo
7-3-1 Hongo, Bunkyo-ku, Tokyo, 113-8656 JAPAN
ayumu@hongo.wide.ad.jp, hiroshi@wide.ad.jp

[†]Information Technology Center
The University of Tokyo
2-11-16 Yayoi, Bunkyo-ku, Tokyo, 113-8658 JAPAN
daisu-mi@nc.u-tokyo.ac.jp, nakayama@nc.u-tokyo.ac.jp

***Abstract*—This paper introduces a novel honeypot for web application. Recently, web applications have been the target of numerous cyber attacks. In order to catch up new vulnerabilities in the applications, using a honeypot system is a feasible solution. However, there remains difficulty for developing a lure-able, protect-able, and deception-able honeypot for web applications. In this paper, we present an approach in which attackers will be automatically isolated from the real web server to the honey web server. The key features are employing migration techniques to create a virtual machine as a honey web server, making the honeypot to equip the same memory and storage devices of the real systems, and controlling network traffic with OpenFlow in order to isolate honeypots from the real server. This paper also shows our design and implementation of INTERCEPT+, a component of honeypot systems for web applications.**

Keywords—Honeypot, Web application, Live Migration, OpenFlow

I. Introduction

Web applications have become one of the popular targets for cyber attacks. This is due to several reasons; for one, the web applications manage a wide array of information including financial data, medical records, social security, therefore attacks aim at stealing the information [1]. Another reason provided by McAfee [2] is the ease of exploitation of web vulnerabilities, combined with the proliferation of low-grade software applications written by inexperienced developers. According to the report from OWASP [3], web vulnerabilities allow a remote attacker to execute injection attacks, e.g., SQL injection [4] and/or Cross Site Scripting [5] attacks that input malicious codes to pose web hijacking, information leakage, malware infection, and so on.

A web application firewall (WAF) is one of the solutions against attacks which filters suspicious code injection. Generally, it inspects the application layer so it usually comes as an appliance type or as a server module [6]. There are several algorithms for distinguishing suspicious codes from HTTP request. The one is rule-based filtering, which checks the HTTP request with the database of known attack patterns. Besides the protection via blacklisting, WAF usually supports whitelisting; with active whitelisting, the rule set of the WAF describes the exact behavior of the application [7]. Different from the rule-based methods, heuristics-based methods calculate the likelihood of being a malicious code and compare the likelihood with the defined discrimination threshold.

Unfortunately, it still remains challenges while preventing code injection attacks. The rapid development of web applications arises numerous program bugs, the cause of web vulnerabilities. Building a perfect blacklist is therefore tedious work. In the case of the whitelisting, the issue is that it requires WAF operators to maintain rules of the legitimate input and output for the applications. The core issue in heuristics-based methods is detection accuracy. To achieve higher detection accuracy, monitoring injection attacks and analysis of them are necessary.

Herein, we present a server-type, honeypot systems for observing web attacks. Theoretically, honeypots are decoy systems to gather information regarding an attacker, and have deception and luring capabilities. A honeypot offers some services that appear perfectly normal to the attackers, and looks alike as if the system has *honey*, e.g., valuable data. Our motivation is to collect the information related to cyber attacks toward web applications using the honeypot.

Our developed *INTERCEPT+* is a component of honeypot for web applications, and it upgrades our previous work named INTERCEPT [8] which was designed for tailoring a honey web server by creating a perfect copy of the legitimate server with live migration techniques. INTERCEPT+ enables to isolate attackers' traffic with the OpenFlow-based system to the honeypot system created by live migration. Since our honeypot has the perfect copy of the real server, the pair of media access control (MAC) address and IP address is the same as that of the real server. In order to avoid MAC and/or IP address conflicts, INTERCEPT+ induces the attack packets to the honeypot regardless of the addresses.

We show that our developed OpenFlow-based packet forwarder can correctly control the packets from normal users and attackers. We also show that Live-Migration technique used in

TABLE I: Interaction Level

Interaction	low	high
Actual OS / applications	no	yes
Risk	low	high
Operational cost	low	high
Performance	low	high

INTERCEPT+ can make its perfect copy in short time and it makes almost no data corruption.

The rest of paper is organized as follows. Section II briefly explain related work, and section III designs the architecture of the honeypot for web applications. Section IV and V implement INTERCEPT+, our proposed honeypot aspects from the migration-based honeypot and OpenFlow-based packet forwarder, in respectively. Section VI shows the preliminary evaluation of INTERCEPT+ and section VII discusses the missing piece toward the suitable honeypot. We finally summarize our contribution in section VIII.

II. Related Work

Honeypot systems can be categorized into four types, namely (i) high-interaction client-type honeypot, (ii), low-interaction client-type honeypot, (iii) high-interaction server-type honeypot, and (iv) low-interaction server-type honeypot. The type, server or client, means that the honeypot systems work as server or client computer. The key characteristics of the interaction was explained in many articles [9]–[11], and the summary is shown in Table I.

The client-type honeypot systems are used to discover new vulnerabilities in the client side systems, such as client OSes, web browsers and their plugins. The honeypot systems usually crawl suspicious web content, allow malicious content to exploit the system, and observe the attack methodologies. For examples of the type (i), HoneyClient [12], Capture-HPC [13], and Marionette [14] are used to analyze malicious URLs for blacklisting. If a system, which was composed of the latest version of Windows, Internet Explorer and plugins, was compromised while browsing, it can be assumed that there was new vulnerability in the system and the honeypot succeeded to find it. Instead of using the actual systems, the type (ii) honeypot systems such as Honey-C [15] and Monkey-Spider [16] mimic the client OSes, IP stacks and applications. While the attacks target particular applications, it can reduce the risk for compromising. However, the observable information during the attacks tends to be limited in comparison to the high-interaction honeypot, as shown in Table I.

The server-type honeypot systems aim at obtaining new vulnerabilities by monitoring the trials of attack, penetration, and intrusion. In order to collect these events, the honeypot systems need to induce the malicious people to be targeted of the attacks. Since the type (iii) honeypot systems are faced to the risk for compromising, the abilities including containment are necessary. Referring to [17], we will explain the requirements of the honeypot systems. In the case of type (iv), the honeypot systems run tools aiming at emulated vulnerable systems. For example, Nepenthes [18] and Deception Toolkit [19] mimic the vulnerable applications. Unfortunately, there are several tools [20], [21] for identifying the systems remotely, therefore, the attacker are easily aware that their targeted system is honeypot.

III. Design of Honeypot for web application

Our grand goal is to develop a novel honeypot system for web applications. In order to discover new vulnerability of web applications, honeypot systems facilitate to collect new attack methodology.

A. Requirements

Along with our survey described in section II, we define the following abilities that honeypot systems for web application should equip.

- Luring ability
  Due to the nature of honeypot, the system must be attractive for attackers.
- Protection ability
  The system must equip protection capability against the attacks. Containment must be work for preventing the abuse of the systems.
- Deception ability
  The system must have deception capability, in other words, attackers cannot distinguish that their targeted system is honeypot.

In the context of the honeypot systems for web applications, we decide to employ high-interaction server-type honeypot systems. Since web attacks often require to login to web applications before their execution, the low-interaction honeypots are required to emulate many web applications. It might be tedious, because this task is equal to make the clone of the targeted application.

B. Components

Our concept of the honeypot system is shown in Figure 1. In order to meet the luring requirements, it employs the actual web applications. A router normally sends to all web requests to the real web server. However, when suspicious web requests are detected, the router sends to the requests to the honey web server. This can meet the protection requirements hence the attacks must not be harm to the real web server.

The one of the difficult point is that a honey web server must have the same state to a real server. In other words, the honey server is required to have the same memory of the real server regarding to the login state, as well as files of web content including applications. We consider such case that the attacker are required to login to web applications before the attacking. Since the attacker's login process is not different to the normal one, this procedure might be done in the real server. After completing the login procedure, the attacker sends suspicious requests to the web application, and the requests are forwarded to the honey web server by the router. To deal with such requests for the honey web server, the server must know the attacker's login state as same as the real server. Otherwise, the honey server may return some error messages to the attacker; the attacker can have a chance to be aware of that they are quarantined to the honey systems. It cannot meet the deception requirement.

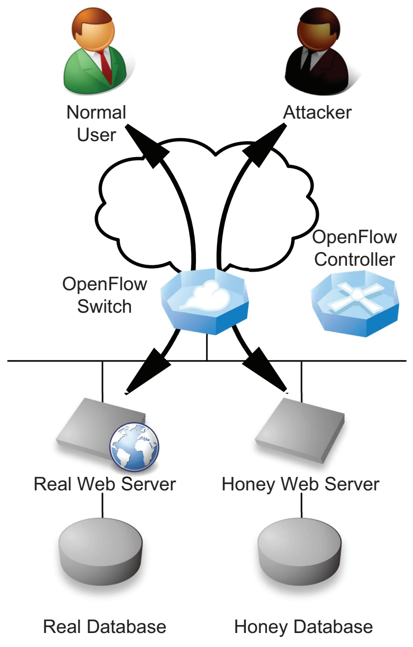


Fig. 1: Concept of High-interaction Server-type Honeypot for web applications

In order to meet the deception requirement, we decided to experiment with migration techniques. They enable a virtual machine to be physically moved from one physical machine to another in a transparent way [22]. It also enables to create the complete memory and storage copy of the real web server.

Another one of the difficult points is address duplication. Since the migrated virtual machine has the same MAC and IP address of the real server, it must be considered for the way of forwarding suspicious web requests to our proposed honeypot. Generally, a network router (L3 router) decides next hop for each packet regarding to the destination IP address, and a network switch (L2 switch) finally decides the destination with the MAC address. Therefore, the conflicts of IP and/or MAC address causes that the suspicious web requests are not delivered to the honeypot.

Instead of the traditional network routers/switches, we decide to employ a Software-Defined Network (SDN) architecture which enables flexible traffic management. Within the architecture, packet forwarding decisions are controlled by the OpenFlow protocol; an OpenFlow Switch (OFS) asks OpenFlow Controller (OFC) to deal with any packets when the OFS attempts to forward the packets. An OFC is an application that manages flow control and dictates how the OFS handles matching packets. As shown in Figure 1, the network switches behind the host OSes of both the real and honey web servers should be capable to the OpenFlow protocol and work as an OFS. For benign requests, the OFS sends them to the real web server. Oppositely, the OFS forwards a request to the honey web server whenever the request seems to be suspicious.

There still remains such problems that *determination of suspicious requests*, *prevention of data leakage*, *avoidance of data corruption*, and *migration in short time period*. For identifying suspicious requests, the likelihood of being an attack, which can be calculated by the heuristics-based WAF, is useful. This is just a case, but if there is slight possibility of the attack, the router in Figure 1 can forward packets to the honey web server. Even if the normal transaction is forwarded to the honey server, it must not be lost because the honey server can observe the transaction. Due to the honey server has the same state of the real server, it must not penalize normal users' convenience.

TABLE II: Specification of Physical Machines

	Sender PC	Receiver PC
CPU	Intel(R) i7-3610QM 2.30 GHz	Intel(R) i5-2450M 2.50 GHz
Memory	8 GB	4 GB
Disk	1 TB ATA	500 GB ATA
NIC	RealTek RTL-8169 (Gigabit Ethernet)	RealTek RTL-8169 (Gigabit Ethernet)

In order to prevent data leakage, it can be considered to separate databases for real and honey web servers. The real database has entire tables and records, and the honey database has limited tables and records that are related to the user who logged in to the web applications. While the honey database should not contain the records to other users, the risk of data leakage might be thwarted.

The missing pieces are *avoidance of data corruption* and *migration in short time period*. This paper also discusses on development of the system in corresponding to these two problems.

IV. Implementation of Live Migration-based Honeypot

In our study, we setup Ubuntu 13.04, Kernel-based Virtual Machine modules to two physical machines; their specifications are shown in Table II. We have modified QEMU [23] source codes. Due to the nature of migration techniques, a migration-source VM will power off to prevent a possible data corruption. Whenever our implement continues to run a source VM after migration, both suppression of the state change in the VM and prevention of the data corruption are necessary.

In order to keep source VM running after migration, we modified the state of source VM will be changed when vm_ stop_force_state() and runstate_set() are called via migration.c. It works fine, so the rest of problem is the data corruption.

At first, we used Full Live Block Migration (FLBM) for preventing data corruption. FLBM supports the entire disk copying, therefore, the source VM and destination VM have respectively their own virtual disk images. It enables that the source and destination VM have the same content in entire memory and block devices as well as avoiding data corruption. However, FLBM requires a lot of time due to the copying full disk images during migration procedure. It might not be good solution for honeypot, because an attacker would feel something is not normal.

Different from FLBM, it was feasible to use of Incremental Live Block Migration (ILBM). In the "incremental" mode, only the blocks that were modified are migrated. QEMU's disk image utility supports to create a snapshot image file. Instead of FLBM, ILBM can dramatically reduce the time during live migration, but it still needs couple of time; it can be assumed that these times are required for verifying disk consistency and for memory migration.

In order to reduce the time for verifying disk consistency, we tested LiveBackup [24] and DriveBackup [25]; the former enables the destination VM periodically polls the source VMs to the data which might be backup, and the latter enables to push the data from the source VM to the destination VM. However, these tools are designed for backup, due to that they only work when the source VM is surely powered off. If the source VM kept running, we confirmed that the data corruption or critical system failure occurred.

Instead of using backup tools, we observed that combination of NFS and advanced multi layered unification filesystem(AUFS) [26] worked fine. AUFS is a implementation of Union File Systems, which provides the function, called branch, to unite several directories into a single virtual filesystem. Based on the solution, we conducted our experiment as follows.

1) Setup snapshot image files
   In order to reduce the further copy-on-write process, we decided to use snapshot. Besides to this, the source physical machine (PM) and the destination PM have the same base image file for the snapshot image file, and use the base image file to the same directory path in the physical machine. For example, if the destination PM puts the base image file into `/root` directory, the source PM also puts the base image file into `/root`.
2) Sharing snapshot
   The source PM creates the snapshot and place the file into NFS server. The destination PM mounts the the NFS server as a read only file systems, and configure AUFS directory. For example, a NFS server exports `/nfs` directory and the destination PM mounts the place as a read-only file system, e.g, `/ronfs`. The PM then mounts `/nfs` as an AUFS system, in which `/ronfs` is a read only directory and any other directory, e.g., `/tmp` is a writable directory.

Instead of ILBM, this methodology employs copy-on-write to keep consistency of the VM disk image files between the source and destination VMs. We observed that the copy-on-write process will be started after the memory migration finished.

Finally, we employ Kemari [27] for reducing the time for memory migration. Kemari provides the feature of the fault tolerance for KVM, and makes the memory migration to be done in the background. The feature also enables the reduction of the time for migrating memory in INTERCEPT+.

Therefore, our prototype implementation is developed by modifying the latest version of Kemari. The figure 2 demonstrates how INTERCEPT+ works for the sender and the receiver VMs. The sender (physical machine) runs the VM as shown in Figure 2a and the receiver also runs the VM as shown in Figure 2b with enabling Kemari's fault tolerant feature. The sender also starts the fault tolerant migration via sender's QEMU monitor console with specifying the IP address of the receiver. In the case of Kemari, the sender VM still runs after finishing migration; the receiver VM does not start, but is suspended. After the receiver VM starts, the sender disables the fault tolerant feature and stops the sender VM by calling `vm_stop(0)` from `migrate_ft_trans_error()` function in `migration.c`. As we mentioned above, we comment out the `vm_stop()` function not to power-off the VM. In addition, we also modify the receiver for remotely starting the VM. Instead of inputting command in the receiver's console, INTERCEPT+ accepts the signal for launching the VM, as shown in Figure 2b.

Algorithm 1 Pseudo Code of the OpenFlow-based Packet Forwarder

```
for all packets do
  Read the list of suspicious IP addresses
  if incoming packets from Internet then
    if the source IP address is listed then
      Send signals to finish live migration
      Forward the packet to the honey web server
    else
      Forward the packet to the real web server
    end if
  else if outgoing packets from the real web server then
    if the destination IP address is listed then
      Discard the packet
    else
      Forward the packet to the Internet
    end if
  else if outgoing packets from the honey web server then
    if the destination IP address is listed then
      Forward the packet to the Internet
    else
      Discard the packet
    end if
  end if
end for
```

TABLE III: Specification of Physical Machines

	OpenFlow PC
CPU	Intel Core i7-3632QM 2.20 GHz
Memory	8 GB
Disk	500 GB ATA
NIC	RealTek RTL8111/8168/8411 (Gigabit Ethernet)

V. Implementation of the OpenFlow-based packet forwarder

This section demonstrates our implementation of OpenFlow-based packet forwarder. We installed the functions of OFC and OFS to one PC, its specification is shown in Table III. We chose RYU [28], a python framework for OFC, to develop our implementation as an OpenFlow application. We also selected Open vSwitch [29] for OFS. The OpenFlow PC has three network interface cards. One is used for the external connection, one is for connecting to the real web server and the other is for the honeypot web server.

The pseudo code of our implementation is shown in

```
# ./qemu-system-x86_64
 -drive file=/nfs/debian-kvm001.qcow2,if=virtio
 -boot d -enable-kvm -monitor stdio -vnc :0
(qemu) migrate -d kemari:tcp:192.168.1.2:4444
```

(a) Sender (192.168.1.1)

```
# ./qemu-system-x86_64
 -drive file=/nfs/debian-kvm001.qcow2,if=virtio
 -boot d -enable-kvm -monitor stdio -vnc :0
 -incoming kemari:tcp:0:4444
# kill -256 (pid)
```

(b) Receiver (192.168.1.2)

Fig. 2: Demonstration of INTERCEPT+ system

Algorithm 1. As we described in section III, it usually forwards all packets from the Internet to the real web server. Besides, all packets from the real server are also forwarded to the Internet. To implement these functions, the forwarder checks the list of suspicious IP addresses for each packet. If the source IP address of the packet was listed, the packet would be forwarded to the honey web server. Oppositely, the implementation continues to work as usual.

We considered such case that the suspicious packets could be detected. The OpenFlow-based packet forwarder immediately ran OS commands for sending signals on the host OS of the honey web server, in order to prepare the honey web server. It then forwarded all packets with suspicious source IP address from the Internet to the honey web server, after live migration was finished. This implementation was also designed to isolate the real server from the attacker. It discarded all packets with the suspicious destination IP address from the real web server to the Internet. In the case of outgoing packets from the honey web server, the forwarder did oppositely.

In addition to that, the forwarder also needed to retrieve the list of the suspicious IP addresses. Our implementation simply read a file for retrieving the address list.

VI. Evaluation

This section provides our evaluation results. Section VI-A shows the performance of the migration-based honeypot, and section VI-B analyzes the latency of the OpenFlow-based packet forwarder.

A. Live Migration-based honeypot

The requirements of the INTERCEPT+ were *avoidance of data corruption* and *migration in short time period*. The section show the detail conditions of the preliminary evaluation in which we used two physical machines as shown in Table II.

This evaluation employed three strategies for launching a honey server, namely Incremental Live Block Migration (ILBM), Live Memory Migration (LMM) with using AUFS for copying block devices, and INTERCEPT+, our modified version of Kemari with AUFS. We also prepared four types of snapshot images, whose size were 20, 50, 100, and 500 MegaBytes (MB) in respectively. Note that 20 MB is the minimum size for the snapshot image file in our experiment.

Based on the above conditions, we measured the time for launching virtual machines. For the cases of LMM with AUFS and INTERCEPT+, we compared the timestamp of the snapshot images created by copy-on-write with the time which started the migration. As for the case of ILBM, we manually measured the turn around time by calculating from the started time and the completed time. The results are summarized in Figure 3a, 3b, 3c, and 3d, where x axis denotes three cases, and y axis denotes the turn around time while creating honeypot. Note that our x axis range for each box graph is limited to the 20 seconds, for readability.

When the disk size was 20 MB, we observed that the minimum average of the turn around time was 2.00 seconds in the case of INTERCEPT+, followed by LMM with AUFS (11.72) and finally ILBM (14.90). In order to compare the responses in a less biased way, we performed Analysis of Variance (ANOVA) and Welch's t-test ($p < 0.05$) for INTERCEPT+ and ILBM, and the result showed that there was statistical difference between the two turn around times in between INTERCEPT+ and ILBM ($p \fallingdotseq 1.40E-32(<0.05), \nu = 18$). In addition to the INTERCEPT+ and LMM with AUFS, there also found statistical difference ($p \fallingdotseq 7.62E-33(<0.05), \nu = 18$).

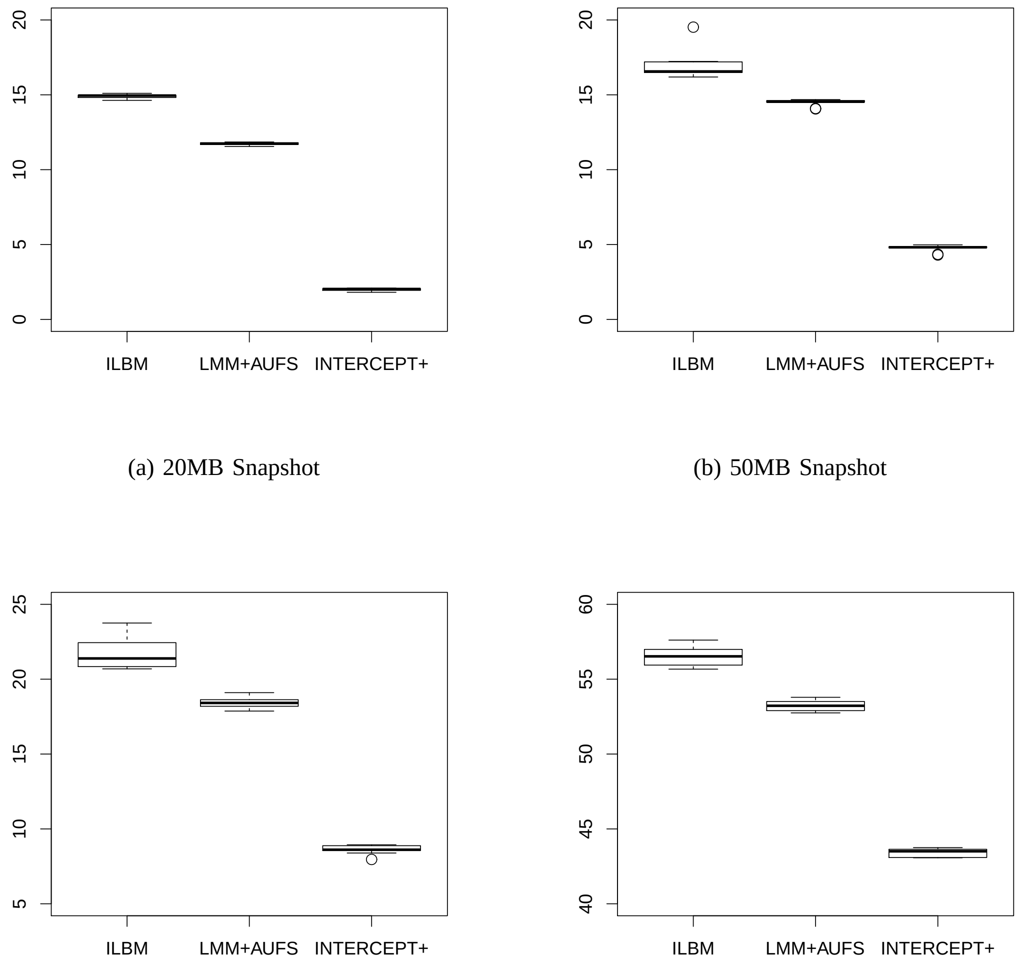
Even if the disk size was 500 MB, the minimum average of the turn around time was 43.23 seconds in the case of INTERCEPT+, followed by LMM with AUFS (53.22) and finally ILBM (56.51). According to our ANOVA results, we performed Student's t-test ($p < 0.05$) for INTERCEPT+ and ILBM and found the statistical difference ($p \fallingdotseq 3.78E-16(<0.05), \nu = 11.86$). There was also statistical difference between INTERCEPT+ and LLM with AUFS ($p \fallingdotseq 4.00E-23(<0.05), \nu = 18$).

We also verified whether or not the data corruption occurred, and observed that there was no data corruption in all cases. Aspect from these observations, INTERCEPT+ succeeded to meet our requirements, *avoidance of data corruption* and *migration in short time period*. However, we also found that the turn around time increases if the snapshot file size became bigger. We will discuss this problem in section VII-B.

B. OpenFlow-based packet forwarder

In this section we performed the results of our preliminary evaluation. We analyzed the latency of the OpenFlow-based forwarder system.

At first we observed the packets under the normal situation in which no attackers exist. Secondly, assuming the IP 10.0.0.2 is detected as an attacker, we observed the packets. The VM migration takes place just when the first packet from 10.0.0.2 arrived. Then the signal is sent to the honeypot and packets from the attacker is sent to the honeypot server. In the last experiment, an attacker is detected and forwarded to the honey



(a) 20MB Snapshot

(b) 50MB Snapshot

(c) 100MB Snapshot

(d) 500MB Snapshot

Fig. 3: The average turn around time in the cases of Incremental Live Brock Migration(ILBM), Live Memory Migration with AUFS (LMM+AUFS), and Kemari-based Live Memory Migration with AUFS (INTERCEPT+).

web server during the TCP session. In every situation, we dumped packets at the client's side. This is because we needed to know whether this honeypot system has ability to hide themselves from attackers, who is at client's side. Of course, we verified that normal users can access to the real web server simultaneously even when the attacker exists and is directed to the honeypot.

The packet dumps taken by Wireshark [30] are shown in Table IV. Table IVa shows the packets from a normal user. In Table IVb the first TCP SYN packet from an attacker triggers the signal to finish live-migration and packets from the attacker is forwarded to the honeypot. In Table IVc, live migration took place during the TCP session. In the Table, the first row from the left shows the time the packets were observed. The second and third rows are the source and destination IP address respectively. The fourth is protocol. This information depends on the expression of Wireshark. The packets for the TCP three way handshake and other packets with ACK flag are TCP, and other packets related to HTTP protocol are shown as HTTP. The last row is the packet information. It shows the SYN, ACK flags about the TCP, while HTTP methods about the HTTP packets.

By comparing the difference between packets from a normal user and an attacker in Table IVa and IVb, we can see a slight delay. The control during TCP session caused one retransmission packet.

As a result, this packet forwarder system can correctly control the attack traffic to the honeypot web server and show almost no strange behaviors. Even though the control during the TCP session causes a retransmission packet, it is still difficult for attackers to detect this honeypot.

TABLE IV: Packets dumps at client's side

(a) Normal Packets

TIME	Source IP	Destination IP	Protocol	Information
0	10.0.0.1	172.16.0.10	TCP	[SYN]
0.0178	172.16.0.10	10.0.0.1	TCP	[SYN, ACK]
0.0179	10.0.0.1	172.16.0.10	TCP	[ACK]
0.0180	10.0.0.1	172.16.0.10	HTTP	GET HTTP/1.1
0.0308	172.16.0.10	10.0.0.1	TCP	[ACK]

(b) Attack Packets

TIME	Source IP	Destination IP	Protocol	Information
0	10.0.0.2	172.16.0.10	TCP	[SYN]
0.0242	172.16.0.10	10.0.0.2	TCP	[SYN, ACK]
0.0243	10.0.0.2	172.16.0.10	TCP	[ACK]
0.0243	10.0.0.2	172.16.0.10	HTTP	GET HTTP/1.1
0.0364	172.16.0.10	10.0.0.2	TCP	[ACK]

(c) Attack Packets (control during TCP session)

TIME	Source IP	Destination IP	Protocol	Information
0	10.0.0.2	172.16.0.1	HTTP	POST
0.0205	172.16.0.1	10.0.0.2	TCP	[ACK]
0.0753	172.16.0.1	10.0.0.2	TCP	Reassembled
0.0753	10.0.0.2	172.16.0.1	TCP	[ACK]
0.0774	172.16.0.1	10.0.0.2	HTTP	HTTP/1.1 OK
0.0774	10.0.0.2	172.16.0.1	TCP	[ACK]
3.7084	10.0.0.2	172.16.0.1	HTTP	GET
3.9285	10.0.0.2	172.16.0.1	HTTP	GET
3.9468	172.16.0.1	10.0.0.2	HTTP	HTTP/1.1 OK
...	...	...	...	...

VII. Discussion

A. Collection of attacks

Our motivation is to collect the information related to cyber attacks toward web applications using the honeypot. This section explains about the content of the information.

As shown in Figure 1, our concept contains detecting a suspicious request, creating a honey web server, preparing honey database, forwarding the request to the created server, and observing the behavior. Imagine if the request can be detected attacks without doubt. When the attack was well known and contained particular phrases that cause injection attacks, the rule-based detection can identify that the request is determinately attack. In this case, the information of the attack is not so new.

Aside from the well known attacks, we want to analyze suspicious attacks with honeypot. As we mentioned in section I, heuristics-based methods calculate the likelihood of being a malicious code and compare the likelihood with the defined discrimination threshold. Assuming if the calculated score 0 means benign and 1 means attacks, and sore 0.5 is the threshold. For example, given calculated score 0.9 (> 0.5), it would be detected as attack rather than benign. However, even if the calculated score is 0.1, it might contain some suspiciousness. By observing the activity during the request, we considered that there is a potential chance for collecting new cyber threats.

Due to the fear of false positive, which is to label benign request as attack, a suspicious request might not be blocked. Instead, our approach seamlessly quarantines the suspicious request to the honey web server. Even if the false positive occurred, the service for the benign user of the web application would be continued without losing any convenience when following two conditions are met: (i) the honey web server has the same memory and disk information of the real server, and (ii) the honey database equips the information that used by this benign user. If the honey database has limited tables and records that are related to the user who logged in to the web applications, as we explained in section III, INTERCEPT+ might not penalize users' convenience even if the false positive error occurred.

B. Deception ability

This section explains about deception ability. We developed INTERCEPT+ to create a perfect copy of environment for web applications based on virtualization techniques. To the best of our knowledge, it would be difficult that honey web server has the same memory of the real web server, including TCP session state and web application session information, without using VM.

The remain issue is time for creating the copy of VMs. If it requires a lot of times, remote attackers will be aware of unusual or strange behavior of the web servers.

Our INTERCEPT+ was designed to reduce the time for migration; use of Kemari and its fault tolerant feature to reduce the time for memory migration, and use of AUFS and its copy-on-write feature for live block migration. The rest of time is the time for copy-on-write in AUFS. The creation time increases when the size of the snapshot disk image file becomes larger. The periodical "re-basing", the procedure of merging the base and the snapshot, might be necessary to keep the size of the snapshot to be reasonable.

The other solution is that use of rapid migration techniques. INTERCEPT+ needs to replicate the VM instance rather than migration, so we need to carefully choose the suitable techniques to avoid data corruption. As well as as Kemari's fault tolerant functions, which do memory migration in the background manner, the fault tolerant functions for block disk image might be feasible. Actually, cloud storage techniques such as GlusterFS [31] and/or Sheepdog [32] have the fault tolerant features and replicate the disk images in the storage network. By modifying their replication functions, there is possibility for creating the perfect copy of VMs in very short time period even if the disk size is large.

Toward development of honeypot for web applications, INTERCEPT+ needs to interconnect to other modules, namely determination modules for suspicious requests. On the integration of INTERCEPT+ and the remaining module, it needs to define the temporal requirements. The required time for detection, direction and preparation of honeypot might be calculated along with each requirement, but it is beyond the scope of this paper.

C. Control of packets

In this section, we discuss remaining issues of the OpenFlow-based packet forwarder. As we mentioned in section III, all IP packets are controlled by the OpenFlow protocol. The packets of other protocols such as Address Resolution Protocol (ARP) are treated as follows in this implementation. ARP packets from the external connection is sent to both the real and honey server and ARP packets from the real/honey server are forwarded to the external connection. This may occur conflicts. This makes a possibility that an attacker can be aware of something unusual.

The other issue is caused of our isolation. Since the honey web server is isolated to the real web server and its surrounding systems, and hence, this leads an attacker to feel different. Even if the attacker intends to use this web server as a step stone of other attacks, our isolation prohibits the attacker to launch another attack from the isolated server.

The way for dealing with these problems are beyond the work of this paper. In these problems, we assumed that the attacker can successfully gain control for injecting OS commands. In this case, it can be regarded that the honeypot usually has already finished its role. From the viewpoint of collecting new vulnerabilities in web applications, these issue were not critical to meet with deception ability.

VIII. Conclusion

The paper introduced INTERCEPT+, a component of the honeypot for web applications. To meet the requirements of the honeypot, we explored the suitable solution and discovered the missing piece, which can avoid the data corruption as well as finishing the migration in short time period. While we employed server-type and high-interaction as a honeypot architecture, the honeypot system created the perfect copy of the actual system. Hence the honeypot had the same memory and disk content of the actual system, web attackers would not be aware of that they were forwarded to honeypot even they checked any TCP and/or web sessions.

We also surveyed several solutions of virtual machine environments and developed our own virtual machine systems. At first, we chose QEMU and modified its source codes, and used the incremental live block migration for creating honey web server. Next, we employed AUFS instead of using incremental live block migration in order to reduce the time for completing live block migration. Further, we modified Kemari and its fault tolerance feature, to reduce the time of the memory migration. We also observed that the creation of the honey web server could be done in few seconds when the size of the snapshot was reasonable.

We finally implemented an OpenFlow-based packet forwarder. Even if our live migration approach arose address duplication, we verified that our implementation could forward the packet to the suitable destination regardless of the duplicated address. We also evaluated its performance and measured the effectiveness. The result showed that we could correctly control the attack and normal packets to our honeypot system.

The rest of work is to improve the performance of our developed INTERCEPT+. There still remains the problem, i.e., the difference of the temporal requirements, but we will develop the suitable honeypot for web applications regarding to the rapid migration techniques in our future work.

Acknowledgment

This research has been supported by the Strategic International Collaborative R&D Promotion Project of the Ministry of Internal Affairs and Communication, Japan, and by the European Union Seventh Framework Programme (FP7/2007-2013) under grant agreement No. 608533 (NECOMA). The opinions expressed in this paper are those of the authors and do not necessarily reflect the views of the Ministry of Internal Affairs and Communications, Japan, or of the European Commission.

References

[1] J. Purcell, "Web Based Attacks," Available at: http://www.sans.org/reading-room/whitepapers/application/web-based-attacks-2053, The SANS Institute, Tech. Rep., 2007.

[2] M. Andrews, "Web Security 101 - introduction," Available at: http://www.mcafee.com/in/resources/audio/foundstone/websec101-intro.html, McAfee, Tech. Rep., 2008.

[3] OWASP, "OWASP Top 10 for 2013 - The Ten Most Critical Web Application Security Risks," The Open Web Application Security Project, Tech. Rep., 2013.

[4] S. McDonald, "SQL Injection: Modes of Attack, Defence, and Why It Matters," Available at: http://www.sans.org/reading-room/whitepapers/securecode/sql-injection-modes-attack-defence-matters-23, The SANS Institute, Tech. Rep., Apr 2002.

[5] S. Cook, "A Web Developer's Guide to Cross Site Scripting," Available at: http://www.sans.org/reading-room/whitepapers/securecode/web-developers-guide-cross-site-scripting-988, The SANS Institute, Tech. Rep., Jan 2003.

[6] I. M. Kim, "Using Web Application Firewall to detect and block common web application attacks," Available at: http://www.sans.org/reading-room/whitepapers/webservers/web-application-firewall-detect-block-common-web-application-attacks-33831, The SANS Institute, Tech. Rep., Nov 2011.

[7] M. Dermann, M. Dziadzka, B. Hemkemeier, A. Hoffmann, A. Meisel, M. Rohr, and T. Schreiber, "Best Practices: Use of Web Application Firewalls," The Open Web Application Security Project, Tech. Rep., 2008.

[8] D. Miyamoto, S. Teramura, and M. Nakayama, "INTERCEPT: High-interaction Server-type Honeypot based on Live Migration," in *Proceedings ofthe 7th International ICST Conference on Simulation Tools and Techniques*, Mar 2014.

[9] A. Mairh, D. Barik, K. Verma, and D. Jena, "Honeypot in Network Security - A Survey," in *Proceedngs of the International Conference on Communication, Computing & Security*, Oct 2011, pp. 600–605.

[10] F. Pouget and T. Holz, "A Pointillist Approach for Comparing Honeypots," in *IEEE Conference on Detection of Intrusions and Malware & Vulnerability Assessment*, Jul 2005.

[11] K. Lin, L. Kyaw, and P. Gyi, "Hybrid Honeypot System for Network Security," *World Academy of Science, Engineering and Technology*, vol. 24, pp. 266–270, 2008.

[12] MITRE, "Honeyclient Project."

[13] The Client Honeynet Project, "Capture-HPC," Avaiable at: http://client-honeynet.org/.

[14] M. Akiyama, M. Iwamura, Y. Kawakoya, K. Aoki, and M. Itoh, "Design and Implementation of High Interaction Client Honeypot for Drive-by-Download Attacks," *IEICE Transactions*, vol. 93, no. B(5), pp. 1131–1139, 2010.

[15] C. Seifert, I. Welch, and P. Komisarczuk, "HoneyC-The Low-Interaction Client Honeypot," Available at http://www.mcs.vuw.ac.nz/cseifert/blog/images/seifert-honeyc.pdf, Victoria University of Wellington, Tech. Rep., 2006.

[16] The Monkey-Spider project, "Monkey-Spider," Availabel at: http://monkeyspider.sourceforge.net/.

[17] L. Spitzner, *Honeypots: Tracking Hackers*, 1st ed. Addison Wesley, 2002.

[18] P. Baecher, M. Koetter, T. Holz, and M. Dornseif, "The Nepenthes Platform: An Efficient Approach to Collect Malware," in *Proceedings of the 9th International Symposium On Recent Advances In Intrusion Detection*, Sep 2006.

[19] F. Cohen, "Deception Toolkit," Available at: http://www.all.net/dtk/index.html.

[20] Nmap, "Free security scanner for network exploration & security audits," Available at: http://nmap.org/.

[21] M. Zalewski, "p0f," Available at: http://lcamtuf.coredump.cx/p0f.shtml.

[22] C. Clark, K. Fraser, S. Hand, J. G. Hansen, E. Jul, C. Limpach, I. Pratt, and A. Warfield, "Live Migration of Virtual Machines," in *Proceedings of the 2nd Symposium on Networked Systems Design and Implementation*, May 2005.

[23] F. Bellard, "QEMU, a Fast and Portable Dynamic Translator," in *Proceedings of the USENIX Annual Technical Conference*, Apr 2005, pp. 41–46.

[24] J. Sundar, "Livebackup - A Complete Solution for making Full and Incremental Disk Backups of Running VMs," Available at: http://wiki.qemu.org/Features/Livebackup.

[25] K. Wolf, "block: drive-backup live backup command," Available at: http://lists.nongnu.org/archive/html/qemu-devel/2013-06/msg04448.html.

[26] J. R. Okajima, "Advanced multi layered unification filesystem," Available at: http://aufs.sourceforge.net.

[27] K. Ohmura and S. Moriai, "Kemari Project," Available at: http://www.osrg.net/kemari/.

[28] Nippon Telegraph and Telephone Corporation, "Ryu Network Operating System," Available at: http://osrg.github.com/ryu, 2012.

[29] Open vSwitch, "Production Quality, Multilayer Open Virtual Switch," Available at: http://openvswitch.org.

[30] G. Combs, "Wireshark, Go Deep," Available at: https://www.wireshark.org/about.html, 2006.

[31] Gluster, Inc., "GlusterFS," Available at: http://www.gluster.org.

[32] Sheepdog Project, "Sheepdog Project," Available at: http://sheepdog.github.io/sheepdog.